Inferix's Proof of Rendering Algorithm

Inferix team

July 16, 2024

Abstract

This document presents the Active Noise Generation and Verification algorithm employed to verify image and video outputs of Inferix distributed graphics rendering service. This algorithm is the first effort introducing asymmetric watermarks into pre-rendering graphics scenes, it is different from classical watermarking method which can only work on media data.

Contents

Chapter 1

Introduction

Inferix is a decentralized GPU network for visual computing and AI, it is built to bridge the needs of users and hardware owners. Its solution meets real-world problems across a range of industries, not only for the AI field but also for high-quality rendering needs. Users (e.g. 3D graphics artists, game developers, enterprises) who need GPU computing power for rendering high-quality graphics can use Inferix system to continuously access these precious resources with faster processing time and more efficient spending. Owners of

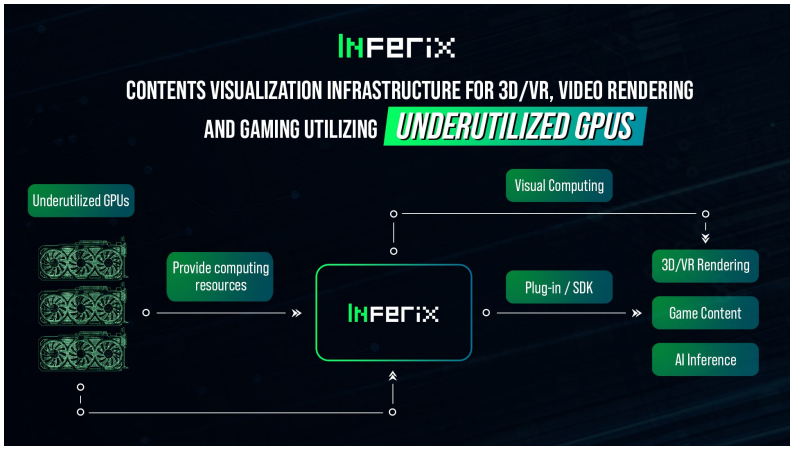


Figure 1.1: High level architecture

GPUs can share idle resources to Inferix network and earn long-term passive income while simultaneously balancing their main jobs or leisure activities.

In this document, we focus mainly on the internal details of Inferix 3D graphics rendering network.

1.1 Rendering network

The graphics rendering service consists in a network of physically decentralized machines called *nodes* which are of 3 kinds: *manager*, *worker* and *verifier*. The *managers* and *verifiers* are dedicated machines of Inferix while the *workers* are machines joined of GPU owners. The number of *workers* is normally much larger than the number of *managers* and *verifiers*. A typical rendering session

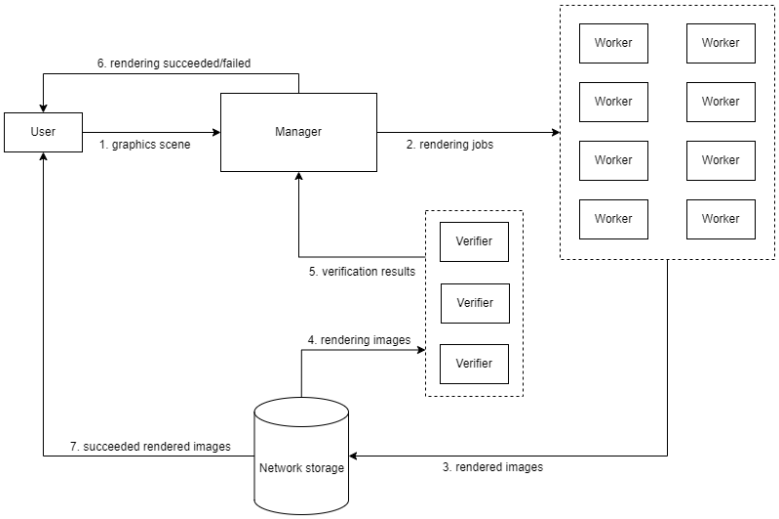


Figure 1.2: Graphics rendering flow

is shown in Figure 1.2:

1. A user submits a graphics scene to some *manager* using the Inferix plugin for client.

2. Receiving the graphics scene, the *manager* builds corresponding rendering jobs consisting of several parameters (range of rendered images, format of the rendering ouput, ...), then pushes these jobs into a rendering job queue. It maintains a pool of *workers*, jobs from the job queue are dispatched to workers in the pool.

3. Receiving a rendering job, a *worker* renders the attached graphics scene using the parameters given by the job. When the rendering finishes, it sends the rendered images to a storage then notifies the *manager*

4.

directly using the Inferix plugin for Blender (or via some Web interface!?). The *manager* creates a job corresponding to this graphics scene, queues this job into a pool then dispatches to *workers* which do the rendering. The rendered images will be sent back from *workers* to several *verifiers* which check if the

images are valid. The validity is notified to the *manager* to accept or reject the rendered results: a job is considered valid only if all results of this job pass the verification, in this case the results will be sent back to users.

(...add a figure here describing the high-level architect of rendering service)

1.2 Output verification

In the network, *workers* are mostly workstations (beside Inferix's dedicated servers) joined by users which want to make profit from their unused computing resources. There is no control (except initial hardware requirements to join the network!?) over these machines, so the graphics rendering is proceeded without being controlled. In order to guarantee that the graphics scenes are correctly rendered, we propose a mechanism called *Active Noise Generation and Verification* which is presented in detail in chapter 2.

(...add a concrete example about an attack)

Chapter 2

Scene watermarking

Before being sent back to the users, the output images of *workers* must be checked if they are actually the results of the corresponding submitted graphics scene. By architectural design, Inferix gains no control whatsoever on *workers* (see 1.2), the graphics rendering softwares installed on *workers* always suffer from risks of being completely analyzed, reversed and maliciously modified. In the long run, one cannot make any assumption about the security on the *worker*'s side, even worse *workers* should be considered adversaries who exhaustively try to bypass any security enforcement applied on their side.

A public-key cryptography approach is using a scheme of *fully homomorphic encryption* (FHE) [Gen09]. The graphics scene is encrypted first by a private key before sending to *workers*. Given the corresponding public key, the homomorphic encryption software performs the graphics rendering on the encrypted scene without needing to decrypt it. Finally, the encrypted rendered results are returned and decrypted at the user's side using the private key. The advantage of FHE is that the *workers*, even can modify the FHE softwares on their side, cannot interfere the FHE graphics rendering processes without being detected. Unfortunately, this approach is impractical since all state-of-the-art implementations will make the performance of the homomorphic encryption graphics rendering become unacceptable [Mar+22].

To handle this problem, we use a watermarking [Cox+97; CMM99] method called *Active Noise Generation and Verification* (ANGV) which is a simpler variant of *proof of ownership* [Cra+97] schemes. *First*, the detector can always judge that the rendered digital content is the result of a valid rendering process whose input is the submitted graphics scene. *Second*, though ANGV needs to modify the initial submitted scene, as a sequence the rendered output will be distorted also, the distortion is regulated to below the human perception capability, so there is no quality degradation. *Third*, the embedded watermark is robust under rendering enhancements (e.g. anti-aliasing) and post-processing operations. *Finally*, there is no need to use a special rendering software on *workers* as in the case of FHE, the performance of graphics rendering will be

not affected.

2.1 Active noise generation and verification

The user submits some Blender work to the *manager*, this work consists of several graphics scenes. Each scene contains information about graphical objects, the camera, light sources and materials [Ble24]. The photorealistic rendering is a sophisticated computation process that calculate light properties at surfaces of all visible objects, results in 3D rendered images of the scene [Hug+14].

To check whether an image output from a *worker* is actually the result of a valid graphics rendering process whose input is a given scene, invisible watermarks [Cra+97] will be embedded into the rendered images. A *verifier* which knows the watermarks It might be worth noting that, different from the context of traditional digital watermarking schemes [Cox+08], these images are not under our control, they are instead output of *workers*. Hence, it is not possible to embed directly watermarks into images, our approach is to embed watermarks into the graphics scene submitted by users before sending it to *workers*.

High level description

The *Active Noise Generation and Verification* (ANGV) consists of two main procedures as described below.

Watermark insertion In practice, a graphics scene may have multiple frames and each worker may render only a subset of these frames. For the simplification purpose, we assume in this section that a scene has only one frame, so the output rendered image is determined uniquely by the scene.

Let $\mathcal{R}$ and S denote respectively the rendering function and the scene, the rendered image I without watermark will be:

$$I = \mathcal{R}(S)$$

It is important to note that I is actually never computed (neither by the *manager* in the watermark insertion procedure nor by *workers* in rendering processes), the equation above represents only an equality. Similar with traditional schemes [CMM99; Cox+97; Cra+97], a watermark W is a sequence of atomic watermarks:

$$W = \{w_1, \ldots, w_n\}$$

where w_i is chosen from some probability distribution and w_i may also depend on S (see section 2.1). Using W and S, we next generate a representative key:

$$K_{\texttt{repr}}(W, S) = \{k_1, \ldots, k_n\}$$

that will be used later for the watermark verification procedure. We do not insert W into the image I but into the scene S using an encoding function $\mathcal{E}$, to create a watermarked scene:

$$\hat{S} = \mathcal{E}(S, W)$$

Finally $\hat{S}$ is sent to *workers* for rendering, that results in a watermarked image:

$$\hat{S} = \mathcal{R}(\hat{S})$$

Watermark verification Given an image $\hat{I}$ and a representative key $K_{\texttt{repr}}$, we first try to recover a watermark $\hat{W}$ from $\hat{I}$ using a decoding function $\mathcal{D}$:

$$\hat{W} = \mathcal{D}(\hat{I}, K_{\texttt{repr}})$$

Next $\hat{W}$ is compared against W, if the difference is above some threshold T

$$\|\hat{W} - W\| \geq T$$

then $\hat{I}$ will be accepted otherwise rejected.

Rendering result In case of being accepted, the rendered image sent back to users is $\hat{I} = \mathcal{R}(\hat{S})$ but not $I = \mathcal{R}(S)$. The encoding function $\mathcal{E}$ and the watermark W are designed so that the distortion of $\hat{I}$ against I is under human perception capability, then $\hat{I}$ can be authentically used as a result of the graphics rendering.

Performance The functions $\mathcal{E}$ and $\mathcal{D}$ are designed so that their computation resource consumption is negligible in comparison with the graphics rendering function $\mathcal{R}$; and the watermark verification does not require I, only $\hat{I}$ is rendered. That means the only overhead caused by ANGV is by $\mathcal{E}$ and $\mathcal{D}$, hence there is no significant effect on the performance of the graphics rendering.

Implementation

In this section, we present in detail the current Inferix's implementation of watermark insertion and verification.

Structure of watermark As described above, a watermark W consists of a sequence of n atomic watermarks, each watermark w_i is a rectangle image of special pattern (see Figure 2.1) of noise chosen from a normal distribution $w_i \sim \mathcal{N}(0, \sigma_i^2)$. The number n is one of the factors decides the robustness of watermark, the higher this number the lower the false positive of watermark verification. Experimentally, W contains about 8 to 10 atomic watermarks.

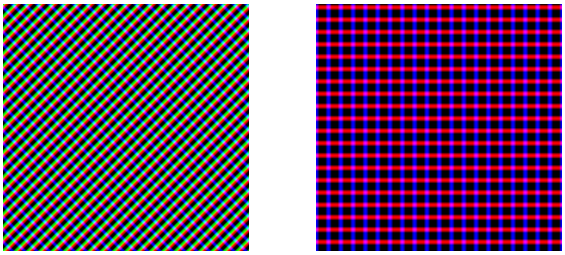


Figure 2.1: Some atomic watermark patterns

Watermark insertion Given some graphics scene S, we embed the watermark W into S by wrapping each atomic watermark as a graphics objects, then insert these objects into S so that they contribute to the result rendered image (i.e. they distort this image).

By the nature of the graphics rendering, an object in the scene will not be visible if there is no light scattered from the surface of the object to the camera. This may be caused by several reasons: the object is not located in the frustum of the camera, is hidden by some other objects, or the material of the object is transparent, etc. The watermark insertion must satisfy first several constraints:

- there are no collisions between watermark objects,
- all watermark objects are located completely in the camera frustum,
- no watermark object is hidden by another object (including both watermark objects and existing objects of the scene).

Representative key generation An atomic watermark w_i contributes to the image $\hat{I}$ as a rectangular region at the location k_i (for that, the rotation vector of the graphics object w_i is kept to be equal with the one of the camera in S). The location k_i on $\hat{I}$ can be precisely calculated without rendering:

$$k_i = \left(x_i^{\mathtt{ul}}, y_i^{\mathtt{ul}}, x_i^{\mathtt{lr}}, y_i^{\mathtt{lr}}\right)$$

where $\left(x_i^{\mathtt{ul}}, y_i^{\mathtt{ul}}\right)$ (resp. $x_i^{\mathtt{lr}}, y_i^{\mathtt{lr}}$) are the upper left (resp. lower right) coordinates of the region. So the insertion of W into S generates a key

$$K_{\mathtt{repr}} = \{k_1, \ldots, k_n\}$$

Other constraints It might be worth noting that k_i represents the distortion caused by the atomic watermark w_i to the rendered image, normally k_i is much smaller than the original size of w_i. On the one hand k_i must not be negligible otherwise the watermark verification is not possible. On the other

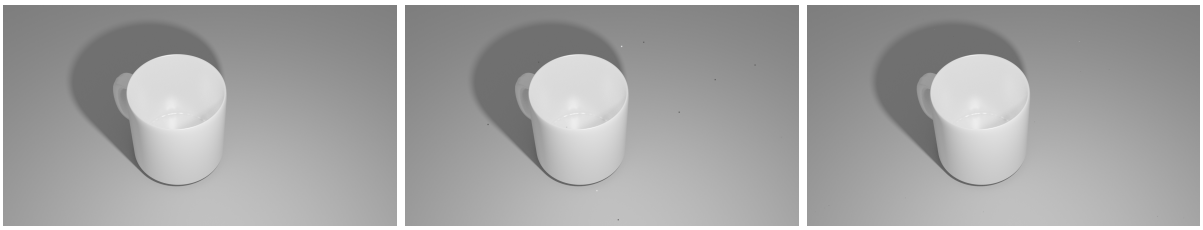
hand, k_i must be kept under the human perception capability. Experimentally we keep a trade-off:

$$\begin{aligned} 2 \leq x_i^{\mathtt{lr}} - x_i^{\mathtt{ul}} \leq 3 \\ 2 \leq y_i^{\mathtt{lr}} - y_i^{\mathtt{ul}} \leq 3 \end{aligned} \tag{2.1}$$

for all $1 \leq i \leq n$ (see Figure 2.2 and Figure 2.3).

(a) Original (b) Watermarked with atomic watermark size = 5 (c) Atomic watermark size = 2

Figure 2.2: Scene watermarked with different size of atomic watermarks: the distortion is observable when the width and the height of an atomic watermark are about 5, but unobservable when these lengths are 2.



(a) Original (b) Atomic watermark size = 7 (c) Atomic watermark size = 3

Figure 2.3: Another scene watermarked with different sizes of atomic watermarks

Watermark verification The verification procedure needs only the representative key $K_{\mathtt{repr}}$ and the rendered image $\hat{I}$. It is not always possible to restore the atomic watermark $w_i \in W$ since most information of w_i is lost after the graphics rendering process (note that k_i is much smaller than the

size of w_i). However, it is neither necessary to restore w_i, instead to verify the existence of W, checking the existence of w_i on the image region determined by k_i on $\hat{I}$ for $1 \leq i \leq n$ is sufficient.

For each region of coordinates $k_i = \left(x_i^{\texttt{ul}}, y_i^{\texttt{ul}}, x_i^{\texttt{lr}}, y_i^{\texttt{lr}}\right)$, we pick a bound region of coordinates:

$$b_i = \left(x_i^{\texttt{ul}} - \delta_i^x, y_i^{\texttt{ul}} - \delta_i^y, x_i^{\texttt{lr}} + \delta_i^x, y_i^{\texttt{lr}} + \delta_i^y\right)$$

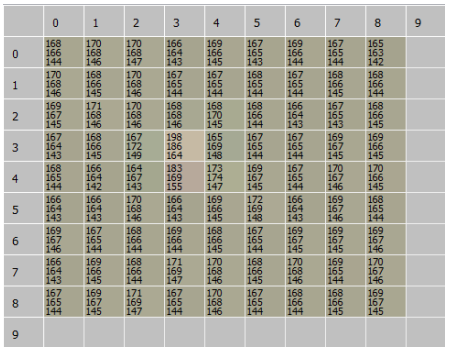
where δ_i^x and δ_i^y are the width and the height of k_i (see Figure 2.4a):

$$\delta_i^x = x_i^{\texttt{lr}} - x_i^{\texttt{ul}} \qquad \delta_i^y = y_i^{\texttt{lr}} - y_i^{\texttt{ul}}$$

To check the contribution of the atomic watermark at the region of coordinates k_i, we apply an edge detection filter (see Figure 2.4b) on the bound region, then compare the mean m_b of the bound region against the mean m_k of the atomic watermark located inside. Experimentally, we will check if

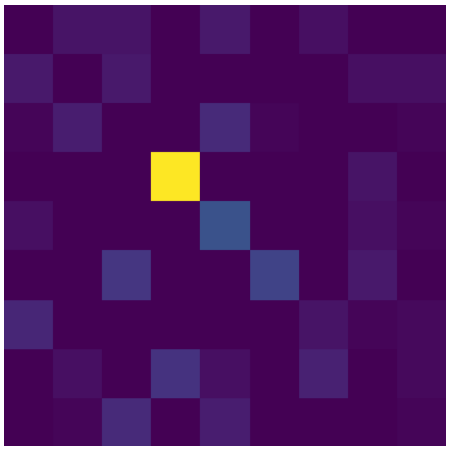
$$m_k - m_b \geq 5 \tag{2.2}$$

to validate the existence of the atomic watermark. This check is proceeded for all n atomic watermarks, if all of them are validated then the rendered image $\hat{I}$ is accepted otherwise rejected.



	0	1	2	3	4	5	6	7	8	9
0	168 166 144	170 168 146	170 168 147	166 164 143	169 166 145	167 165 143	169 166 144	167 165 144	165 163 142	
1	170 168 146	168 166 145	170 168 146	167 165 144	167 165 144	168 165 144	167 165 144	168 166 145	168 166 144	
2	169 167 145	171 168 146	170 168 146	168 168 146	168 170 145	168 166 144	166 164 143	167 165 143	168 166 145	
3	167 164 143	168 166 145	167 172 149	198 186 164	165 169 148	167 165 144	167 165 144	169 167 145	169 166 145	
4	168 165 144	166 164 142	164 167 143	183 169 155	173 174 147	169 167 145	167 165 144	170 167 146	170 166 145	
5	166 164 142	166 164 143	170 168 146	166 164 143	169 166 145	172 169 148	166 164 143	169 167 146	168 165 144	
6	169 167 146	167 165 144	168 166 144	169 166 144	168 166 145	167 165 144	169 167 145	169 167 145	169 167 146	
7	166 164 143	169 166 145	168 166 144	171 169 147	170 168 146	168 166 145	170 168 146	169 165 144	170 167 146	
8	167 165 144	169 167 145	171 169 147	167 165 144	170 168 146	167 165 144	168 166 144	168 166 145	169 167 145	
9										

(a) Pickup bound region of size 9×9 pixels, the atomic watermark region of size 3×3 located at the center of the bound (numbers on each pixel are RGB color values)



(b) Pickup bound region after an edge detection filter

Figure 2.4: Watermark verification

Robustness We now discuss several details about the contribution of the watermark on the rendered image. There are two related constraints:

- the side trade-off in Equation 2.1 is used to keep the rendering distortion under the human perception capability, and
- the mean difference in Equation 2.2 is used to validate the existence of an atomic watermark

which are unfortunately disproportional: if the side is too small

Performance

2.2 Threat analysis

Bibliography

[Ble24] Blender. *Blender Reference Manual*. 2024. URL: `https://docs.blender.org/manual/en/latest/`.

[CMM99] Ingemar J. Cox, Matthew L. Miller, and Andrew L. McKellips. "Watermarking as Communications with Side Information". In: *Proceedings of the IEEE* 87.7 (July 1999).

[Cox+08] Ingemar J. Cox et al. *Digital Watermarking and Steganography*. Ed. by Morgan Kaufmann. Elsevier, 2008. ISBN: 9780123725851.

[Cox+97] Ingemar J. Cox et al. "Secure Spread Spectrum Watermarking for Multimedia". In: *IEEE Transactions on Image Processing* (Dec. 1997).

[Cra+97] Scott A. Craver et al. "Can Invisible Watermarks Resolve Rightful Ownerships?" In: *Storage and Retrieval for Image and Video Databases*. Ed. by Ishwar K. Sethi and Ramesh C. Jain. SPIE, Jan. 1997.

[Gen09] Craig Gentry. "Fully Homomorphic Encryption using Ideal Lattices". In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing* (May 2009).

[Hug+14] John F. Hughes et al. *Computer Graphics: Principles and Practice*. 3rd. Addison-Wesley, 2014. ISBN: 0321399528.

[Mar+22] Chiara Marcolla et al. "Survey on Fully Homomorphic Encryption, Theory, and Applications". In: *Proceedings of the IEEE* 110.10 (2022).